The Grammar of Graphics

Martin Frigaard

2021-05-01

Contents

The ggplot2 package is one of most commonly used tools for data visualizations. For more on the grammar, see the online text titled, ggplot2: Elegant Graphics for Data Analysis. If you're looking for a cookbook (graphs and code to build them), see the R Graphics Cookbook, 2nd edition.

0.1 Why have a grammar of graphics?

Wilhelm von Humboldt has described a language as a system for "*making infinite use of finite means.*" Grammar is the way we convert the thoughts in our heads into discrete concepts (i.e. words), and then we apply a set of rules (or syntax) to create and display comprehensible statements (for humans or computers).

In this sense, ggplot2 gives us an ability to communicate the complexities of any data visualization in the same way that any specialized vocabulary allows us to precisely and unambiguously define ideas.

0.2 Import the data

We will begin by importing the data from the wrangling section. These data come from a wikipedia table on the deployment of COVID-19 vaccinations. The code below will scrape the html table and store the results in `COVID19VaxDistByLoc`.

0.2.1 Scrape wikipedia

1. We load the `tidyverse`, `rvest`, and `xml2` packages with `library()`

2. The url for the wikipedia page is read into R with `xml2::read_html()` and stored in `wiki_html` as a list containing `xml_document` and `xml_node`

3. The `rvest::html_nodes()` function looks for a CSS `"table"` in `wiki_html` and stores these in `wiki_html_tables`

4. We use the bracket subsetting (`[]`) and `base::grep()` to find tables with the word `"distribution"` in them and store these in `relevant_tables`

5. Now we can use the `rvest::html_table()` function to 'harvest' the tables stored in the first position of `relevant_tables` and set the `fill` argument to `TRUE` (`[[1]]`), and store the output in `COVID19VaxDistByLoc`.

6. The `COVID19VaxDistByLoc` is a rectangular `data.frame` object, but we only want the first three columns (`[ , 1:3]`), and we want to rename these `"location"`, `"n_vaccinated"`, and `"perc_of_pop"`.

```r
# packages -----------------------------------------------------------
library(tidyverse)
library(rvest)
library(xml2)

# scrape wikipedia table ---------------------------------------------
# Read html from url
wiki_html <- xml2::read_html("https://en.wikipedia.org/wiki/COVID-19_vaccine")
# extract html nodes
wiki_html_tables <- wiki_html %>% rvest::html_nodes(css = "table")
# identify relevant html table with 'distribution' in the title
relevant_tables <- wiki_html_tables[grep("distribution", wiki_html_tables)]
# convert table to data.frame
COVID19VaxDistByLoc <- rvest::html_table(relevant_tables[[1]],
                                         fill = TRUE)
# assign names to first three columns
COVID19VaxDistByLoc <- COVID19VaxDistByLoc[ , 1:3] %>%
  magrittr::set_names(x = ., value = c("location", "n_vaccinated",
                                       "perc_of_pop"))
glimpse(COVID19VaxDistByLoc)
```

```
## Rows: 198
## Columns: 3
## $ location     <chr> "World[d]", "China[e]", "United States", "India", "EU", "~
## $ n_vaccinated <chr> "595,234,872", "265,064,000", "144,894,586", "125,376,952~
## $ perc_of_pop  <chr> "7.6%", "--", "43.3%", "9.1%", "23.9%", "50.4%", "13.7%",~
```

0.2.2 Date-stamp and export

This is a good time to export these data into the `data/raw` folder (in case the numbers change the next time we scrape the table).

```r
readr::write_csv(x = COVID19VaxDistByLoc,
                 file = paste0("../data/raw/",
                               base::noquote(lubridate::today()),
                 "-COVID19VaxDistByLoc.csv"))
# verify
fs::dir_tree("../data/raw/", regexp = "COVID19")
```

```
## ../data/raw/
## +-- 2021-04-15-COVID19VaxDistByLoc.csv
## +-- 2021-04-17-COVID19VaxDistByLoc.csv
## +-- 2021-04-21-COVID19VaxDistByLoc.csv
## +-- 2021-04-22-COVID19VaxDistByLoc.csv
## +-- 2021-04-23-COVID19VaxDistByLoc.csv
## +-- 2021-04-26-COVID19VaxDistByLoc.csv
## +-- 2021-04-27-COVID19VaxDistByLoc.csv
## \-- 2021-05-01-COVID19VaxDistByLoc.csv
```

We can see these data have been downloaded on multiple days (starting on `2021-04-15`)

0.3 Look at the data

> "A problem well-defined is a problem half solved." John Dewey

Before we start any data wrangling, we need to look at the data in it's 'natural state.' Viewing the data gives us an opportunity to quantify the catastrophe we're dealing with, and let's us plan a path forward.

There are multiple functions for looking at your data in R. I like to start with the **`utils::head()`** and **`utils::tail()`** functions see the 'top' and 'bottom' a dataset.

`utils::head()` shows us the top six rows of `COVID19VaxDistByLoc`:

```
utils::head(COVID19VaxDistByLoc)
```

```
## # A tibble: 6 x 3
##   location       n_vaccinated perc_of_pop
##   <chr>          <chr>        <chr>
## 1 World[d]       595,234,872  7.6%
## 2 China[e]       265,064,000  --
## 3 United States  144,894,586  43.3%
## 4 India          125,376,952  9.1%
## 5 EU             106,409,808  23.9%
## 6 United Kingdom 34,216,087   50.4%
```

We can change the number of rows **`head()`** or **`tail()`** returns by supplying a number to the **`n`** argument.

```
utils::head(COVID19VaxDistByLoc, n = 10)
```

```
## # A tibble: 10 x 3
##    location       n_vaccinated perc_of_pop
##    <chr>          <chr>        <chr>
##  1 World[d]       595,234,872  7.6%
##  2 China[e]       265,064,000  --
##  3 United States  144,894,586  43.3%
##  4 India          125,376,952  9.1%
##  5 EU             106,409,808  23.9%
##  6 United Kingdom 34,216,087   50.4%
##  7 Brazil         29,149,512   13.7%
##  8 Germany        22,393,183   26.7%
##  9 Turkey         13,712,254   16.3%
## 10 France         15,254,118   22.4%
```

0.3.1 exercise

Use the `utils::tail()` function below to view the bottom 10 rows of `COVID19VaxDistByLoc`.

```
utils::tail(COVID19VaxDistByLoc, n = __)
```

0.3.2 solution

```
utils::tail(COVID19VaxDistByLoc, n = 10)
```

```
## # A tibble: 10 x 3
##    location                 n_vaccinated              perc_of_pop
##    <chr>                    <chr>                     <chr>
##  1 "South Sudan"            "947"                     "0.0%"
##  2 "Libya"                  "750"                     "0.0%"
##  3 "Nauru"                  "700"                     "6.5%"
##  4 "Armenia"                "565"                     "0.0%"
##  5 "Cameroon"               "400"                     "0.0%"
##  6 "F.S. Micronesia[512]"   "20,423"                  "19.7%"
##  7 "Marshall Islands[512]"  "14,544"                  "24.9%"
##  8 "Palau[512]"             "12,511"                  "69.9%"
##  9 "Vatican City[513][514]" "22"                      "2.7%"
## 10 "Sources\nList of sourc~ "Sources\nList of sources~ "Sources\nList of source~
```

We've covered other functions for viewing your data (`dplyr::glimpse()`, `utils::str()`, and `View()`), and I recommend using any combination of them to get a good understanding of what you're dealing with. We can already see a few of the columns need to be addressed before we can start visualizing, so let's write up a plan for wrangling these variables:

1. The last row in `COVID19VaxDistByLoc` has some metadata (data about data) that needs to be extracted before we can visualize.
2. We need to remove the alphabetic identifier for each country `location` (i.e., `World[d]` and `China[e]`).
3. The number vaccinated variable (`n_vaccinated`) has commas (`,`) and needs to be converted to a number.

4. The percent of population variable (`perc_of_pop`) has symbols (decimals, percent symbols (%), and missing values (`--`)), which is making R treat it as a character, so these will have to be removed.

0.4 Step 1: Remove metadata

We can use the `dplyr::filter` function to remove the last row with the `Sources`. We're going to combine `filter()` with the `stringr::str_detect()` function so we can identify the row with the word 'Sources'. The `stringr` package is part of the `tidyverse` and comes with some excellent functions for manipulating strings (characters).

`stringr::str_detect()` takes a `string` argument, which will be our `location` variable in `COVID19VaxDistByLoc`, and a `pattern` argument, which we will specify as `"Source"`.

```
stringr::str_detect(string = COVID19VaxDistByLoc$location,
                    pattern = "Source")
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [193] FALSE FALSE FALSE FALSE FALSE  TRUE
```

As we can see, only the last row is identified as having the `"Source"` pattern. But what if we want the *opposite* logical values designated? Fortunately, `str_detect()` has a `negate` argument we can set to `TRUE`.

```
stringr::str_detect(string = COVID19VaxDistByLoc$location,
                    pattern = "Source", negate = TRUE)
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [73]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [85]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [97]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [109]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [121]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [145]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [157]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [169]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [181]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [193]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

0.4.1 exercise

Use `str_detect()` and `filter()` to remove the metadata row, and assign the output to `WikiCovid`. Change the `negate` argument to `TRUE` for these data.

```
WikiCovid <- COVID19VaxDistByLoc %>%
  dplyr::filter(str_detect(string = COVID19VaxDistByLoc$location,
                    pattern = "_____", negate = ____))
glimpse(WikiCovid)
```

0.4.2 solution

See solution below.

```
WikiCovid <- COVID19VaxDistByLoc %>%
  dplyr::filter(str_detect(string = COVID19VaxDistByLoc$location,
                    pattern = "Source", negate = TRUE))
glimpse(WikiCovid)
```

```
## Rows: 197
## Columns: 3
## $ location     <chr> "World[d]", "China[e]", "United States", "India", "EU", "~
## $ n_vaccinated <chr> "595,234,872", "265,064,000", "144,894,586", "125,376,952~
## $ perc_of_pop  <chr> "7.6%", "--", "43.3%", "9.1%", "23.9%", "50.4%", "13.7%",~
```

0.4.3 exercise

Now use `str_detect()` with `filter()` to extract the metadata row with the `"Source"` pattern, and assign the output to `WikiCovidSource`. Don't change the `negate` argument this time.

```
WikiCovidSource <- COVID19VaxDistByLoc %>%
  dplyr::filter(str_detect(string = __________________________,
                    pattern = "_______"))
```

0.4.4 solution

See solution below.

```
WikiCovidSource <- COVID19VaxDistByLoc %>%
  dplyr::filter(str_detect(string = COVID19VaxDistByLoc$location,
                    pattern = "Source"))
glimpse(WikiCovidSource)
```

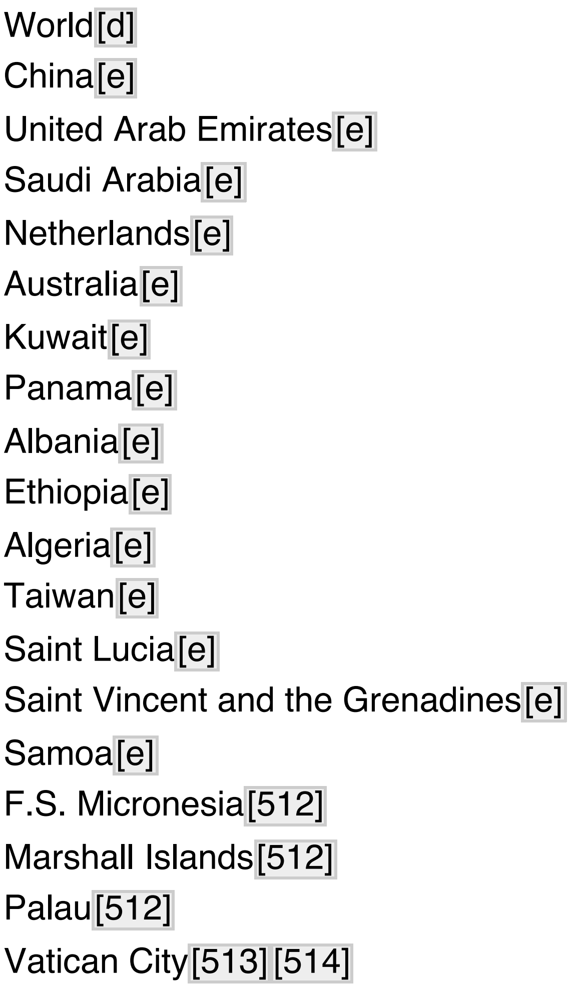
```
## Rows: 1
## Columns: 3
## $ location     <chr> "Sources\nList of sources by country.\nNotes\n\n^ Latest ~
## $ n_vaccinated <chr> "Sources\nList of sources by country.\nNotes\n\n^ Latest ~
## $ perc_of_pop  <chr> "Sources\nList of sources by country.\nNotes\n\n^ Latest ~
```

We changed the name of the `COVID19VaxDistByLoc` dataset to `WikiCovid` so we can differentiate the changed data from the raw data. This is a good practice because you might need to revert back to the original dataset along the way (or view it for comparison).

0.5 Step 2: Remove string characters

For the next step in wrangling the `location` variable, we will want to identify all the letters in brackets using `stringr::str_view_all()`. Below is an example of how it works:

```
str_view_all(string = WikiCovid$location, pattern = "\\[[^\\[\\]]+\\]", match = TRUE)
```



World[d]
China[e]
United Arab Emirates[e]
Saudi Arabia[e]
Netherlands[e]
Australia[e]
Kuwait[e]
Panama[e]
Albania[e]
Ethiopia[e]
Algeria[e]
Taiwan[e]
Saint Lucia[e]
Saint Vincent and the Grenadines[e]
Samoa[e]
F.S. Micronesia[512]
Marshall Islands[512]
Palau[512]
Vatican City[513][514]

`str_view_all()` shows us all the `locations` with a bracket `[]` + letter/number indicator. Don't worry if you don't know what the regular expression pattern (`"\\[[^\\[\\]]+\\]"`) is doing. We will cover regular expressions in a later section (if you can't wait, check out the Strings chapter of R for Data Science). The main takeaway here is that we need to provide a `string` (i.e. the variable name), and a pattern we wish to view.

0.5.1 exercise

Now that we've successfully identified the regular expression pattern for matching all the strings we want to remove, we can use `dplyr::mutate()` and `stringr::str_remove_all()` to remove these numbers and letters from the `location` column:

- copy and paste the `pattern` from the `stringr::str_view_all()` function above into the `pattern` argument for `stringr::str_remove_all()` below:

```
WikiCovid %>%
  mutate(
    # remove bracket indicators ([])
    location = stringr::str_remove_all(string = location,
                                       pattern = "_______________"))
```

0.5.2 solution

Check your solution below:

```
WikiCovid %>%
  mutate(
    # remove bracket indicators ([])
    location = stringr::str_remove_all(string = location,
                                       pattern = "\\[[^\\[\\]]+\\]"))
```

```
## # A tibble: 197 x 3
##    location       n_vaccinated perc_of_pop
##    <chr>          <chr>        <chr>
##  1 World          595,234,872  7.6%
##  2 China          265,064,000  --
##  3 United States  144,894,586  43.3%
##  4 India          125,376,952  9.1%
##  5 EU             106,409,808  23.9%
##  6 United Kingdom 34,216,087   50.4%
##  7 Brazil         29,149,512   13.7%
##  8 Germany        22,393,183   26.7%
##  9 Turkey         13,712,254   16.3%
## 10 France         15,254,118   22.4%
## # ... with 187 more rows
```

0.6 Step 3: Removing commas

The next variable we want to address is the number vaccinated, or `n_vaccinated`. These numbers were formatted with commas in the Wikipedia table (which is common), so R treated them like a character variable. We will use the `readr::parse_number()` to convert `n_vaccinated` to a numerical variable.

0.6.1 exercise

Enter the `n_vaccinated` variable into the `readr::parse_number()` function below:

```r
WikiCovid %>%
  mutate(
    # remove bracket indicators ([])
    location = stringr::str_remove_all(string = location,
                                       pattern = "\\[[^\\[\\]]+\\]"),
    n_vaccinated = readr::parse_number(x = ____________))
```

0.6.2 solution

See solution below:

```r
WikiCovid %>%
  mutate(
    # remove bracket indicators ([])
    location = stringr::str_remove_all(string = location,
                                       pattern = "\\[[^\\[\\]]+\\]"),
    n_vaccinated = readr::parse_number(n_vaccinated))
```

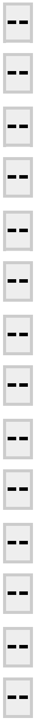
```
## # A tibble: 197 x 3
##    location       n_vaccinated perc_of_pop
##    <chr>                 <dbl> <chr>
##  1 World             595234872 7.6%
##  2 China             265064000 --
##  3 United States     144894586 43.3%
##  4 India             125376952 9.1%
##  5 EU                106409808 23.9%
##  6 United Kingdom     34216087 50.4%
##  7 Brazil             29149512 13.7%
##  8 Germany            22393183 26.7%
##  9 Turkey             13712254 16.3%
## 10 France             15254118 22.4%
## # ... with 187 more rows
```

Now the data are beginning to look wrangled! We only have one more variable to go!

0.7 Step 4: Remove decimals and symbols

The `perc_of_pop` variable poses a few challenges, starting with the `%` symbol. We can remove this with `stringr::str_remove_all()` *or* the `readr::parse_number()` function. But we can also see the missing values are represented with a `--` symbol. We should see how many missing values there are in this dataset using `stringr::str_view_all()`.

```
str_view_all(string = WikiCovid$perc_of_pop, pattern = "--", match = TRUE)
```



```
--
--
--
--
--
--
--
--
--
--
--
--
--
--
```

This is not an insignificant amount! To get an exact count, we can combine `sum()` and `str_detect()`:

```
sum(str_detect(WikiCovid$perc_of_pop, pattern = "--"))
```

```
## [1] 14
```

Now we need to decide how to deal with these missing values *and* the percentage symbols. We will test both `stringr::str_remove_all()` and `readr::parse_number()` below to see which one is best:

```
stringr::str_remove_all(string = head(WikiCovid$perc_of_pop), pattern = "%")
```

```
## [1] "7.6"  "--"   "43.3" "9.1"  "23.9" "50.4"
```

`stringr::str_remove_all` gives us no problems, and returns the original symbol for the missing `China` value (`--`). What about `readr::parse_number()`?

```
readr::parse_number(x = head(WikiCovid$perc_of_pop))
```

```
## [1]  7.6   NA 43.3  9.1 23.9 50.4
## attr(,"problems")
## # A tibble: 1 x 4
##     row   col expected actual
##   <int> <int> <chr>    <chr>
## 1     2    NA a number --
```

`readr::parse_number()` tells us there was a `parsing failure`, and this value has been changed to `NA`. This is preferred because 1) it requires fewer steps, and 2) it will handle other missing values in the future.

0.7.1 exercise

Add the code for removing the percentage symbols from `perc_of_pop` with `readr::parse_number()` to the `mutate()` function below:

```
WikiCovid <- WikiCovid %>%
  mutate(
    # remove bracket indicators ([])
    location = stringr::str_remove_all(string = location,
                                       pattern = "\\[[^\\[\\]]+\\]"),
    n_vaccinated = readr::parse_number(n_vaccinated),
    # add perc_of_pop
    perc_of_pop = readr::parse_number(_____________)
  )
```

0.7.2 solution

See solution below:

```
WikiCovid <- WikiCovid %>%
  mutate(
    # remove bracket indicators ([])
    location = stringr::str_remove_all(string = location,
                                       pattern = "\\[[^\\[\\]]+\\]"),
    n_vaccinated = readr::parse_number(n_vaccinated),
    perc_of_pop = readr::parse_number(perc_of_pop)
  )
```

We can see the message about the parsing failures (which we expected). Let's view our wrangled dataset below:

```
WikiCovid
```

```
## # A tibble: 197 x 3
##    location       n_vaccinated perc_of_pop
##    <chr>                  <dbl>       <dbl>
##  1 World              595234872         7.6
##  2 China              265064000        NA
##  3 United States      144894586        43.3
##  4 India              125376952         9.1
##  5 EU                 106409808        23.9
##  6 United Kingdom      34216087        50.4
##  7 Brazil              29149512        13.7
##  8 Germany             22393183        26.7
##  9 Turkey              13712254        16.3
## 10 France              15254118        22.4
## # ... with 187 more rows
```

0.8 Explore your data

R has thousands of custom built packages for visualizing data. One of the packages we'll be using a lot in this course is `skimr`, which provides a "*A frictionless, pipeable approach to dealing with summary statistics.*"

What is 'pipeable'?

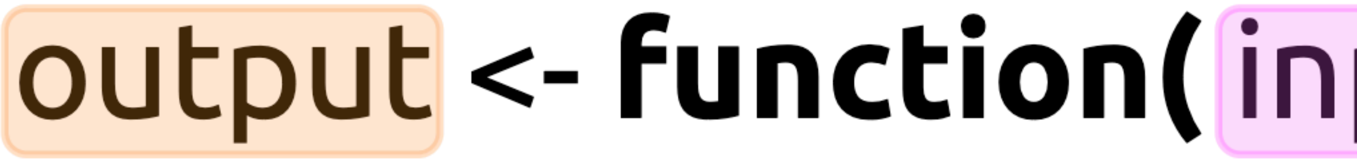
The pipe (`%>%`) from the `magrittr` package is what's referred to as syntactic sugar (yes, that's really a term) because it's,

"*syntax within a programming language that is designed to make things easier to read or to express*"

0.8.1 How pipes work

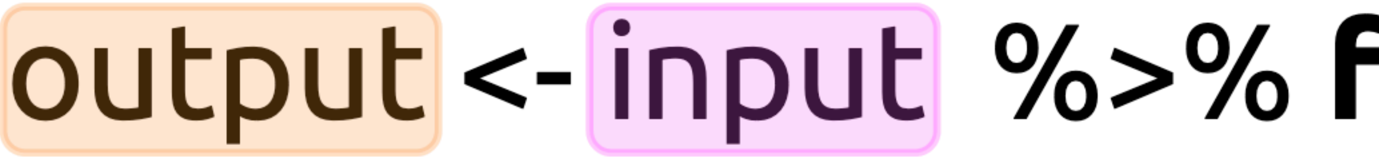
R is a functional programming language. In standard math notation, the common way to write a function is `f(x)` or `y = f(x)`, which is read as "*f of x*" or "*y equals f of x*".

Pipes restructure the function syntax, so this:



```
output <- function(in
```

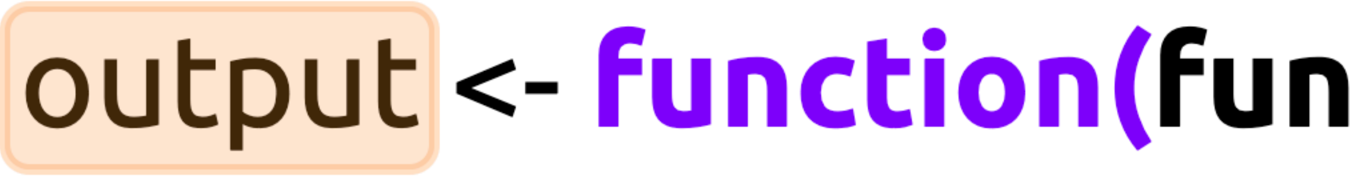
Becomes this:



```
output <- input %>% f
```

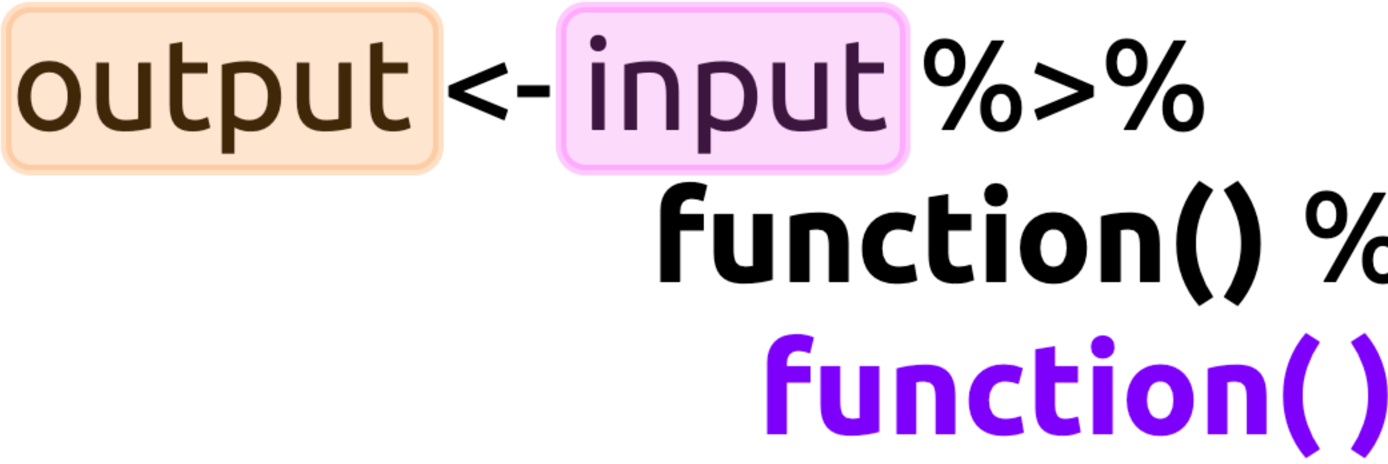
0.8.2 Pipelines

As you can imagine, writing code like this can get complicated if we wanted to use multiple functions (as we typically do), Without the pipe, we have to write these as nested functions (i.e. `h(f(x))`).



```
output <- function(fun
```

With the pipe, we can rewrite this code to the following:



```
output <- input %>%
                function() %
                    function( )
```

Using the pipe makes code easier to 1) think about, 2) write, and 3) read.